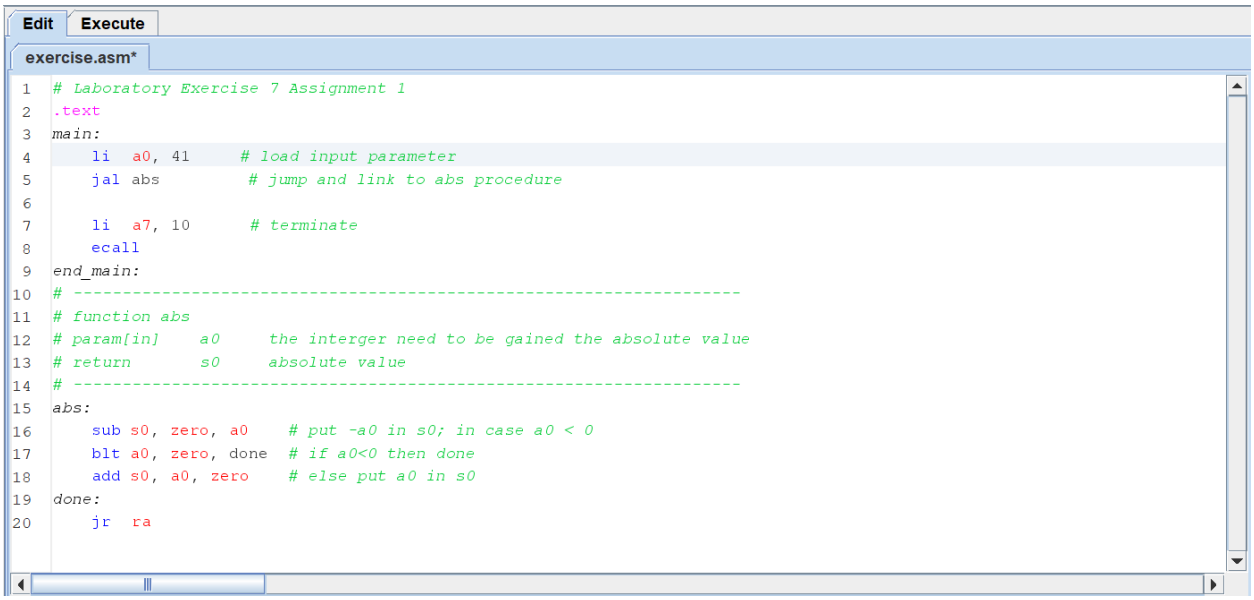
## Student name: Lê Ngọc Anh Vũ

## Student ID: 20236014

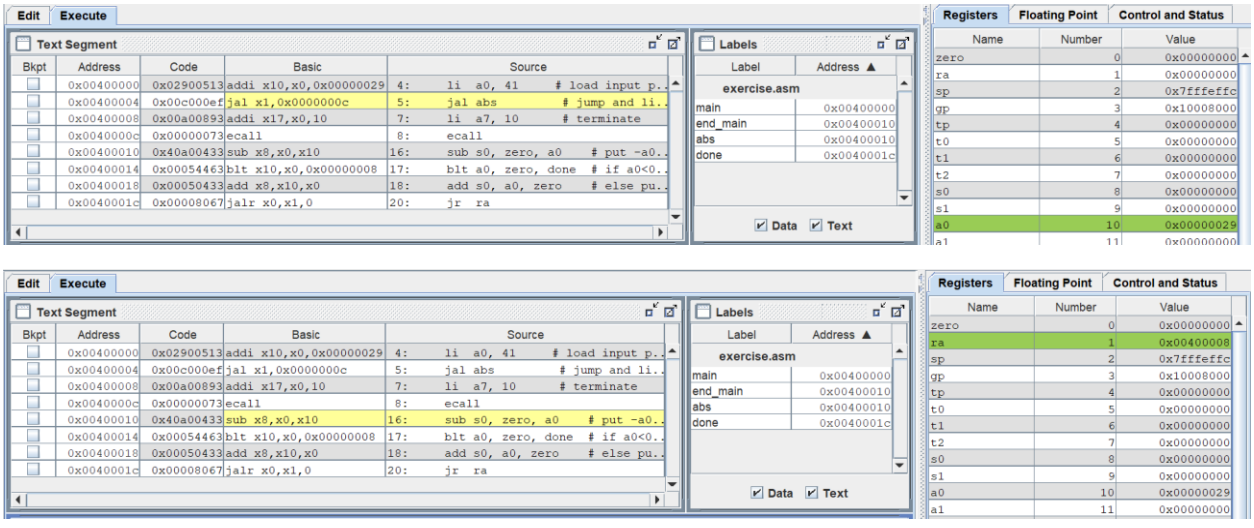
## LAB 7:

**Assignment 1: Create a project to implement Home Assignment 1. Compile and simulate it. Change the program parameters (register a0) and observe the execution results. Run the program in the single-step mode and pay attention to the changes in registers, especially the pc and ra registers.**

We change the value of program parameters (register a0) to 41 (a positive number) as the picture below:



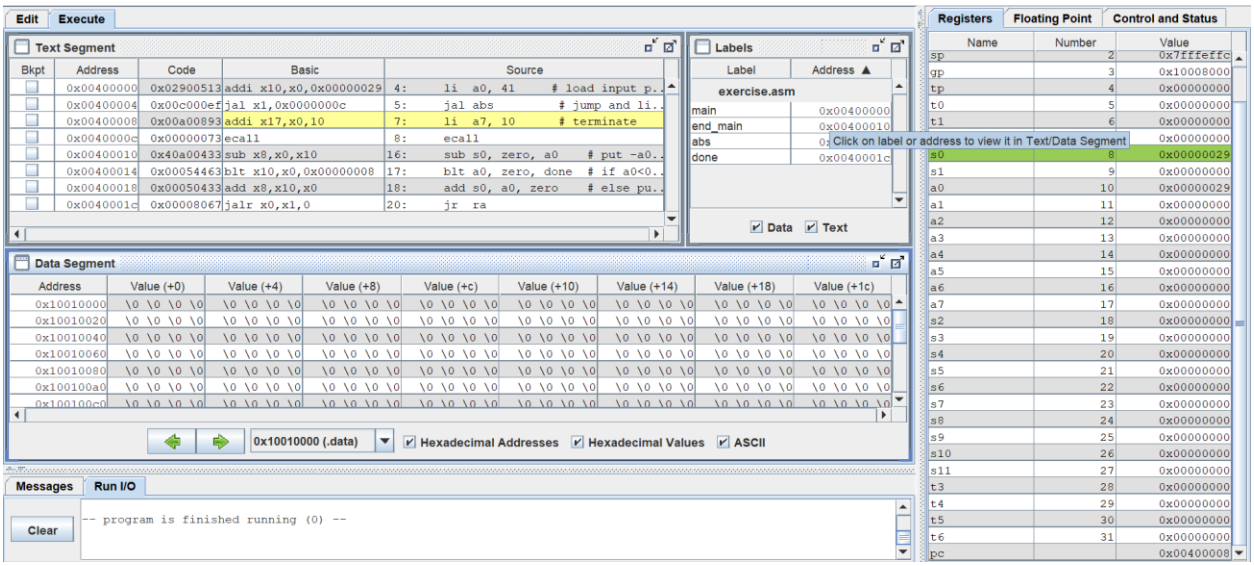

After executing the instruction “jal abs”, value of register ra changes from 0x00000000 to 0x00400008 which is the address of the next instruction “li a7, 10”. The register pc jumps from address 0x00400004 to address 0x00400010



| pc | | 0x00400004 |
|---|---|---|

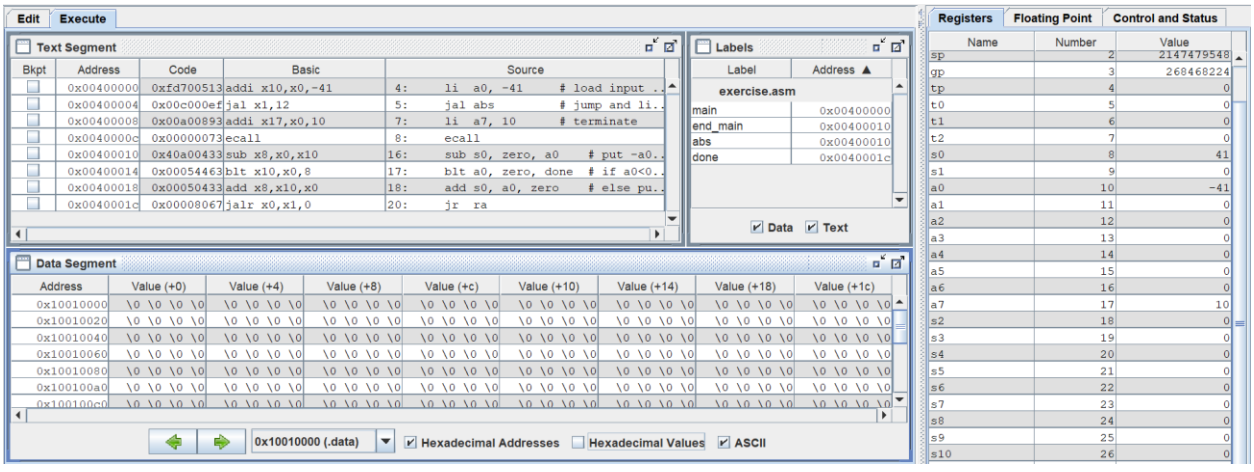
| pc | | 0x00400010 |
|---|---|---|

After executing the instruction “jr ra”, the register jumps again to th address 0x00400008




Then, the program ends with correct result which is s0=41 ( the absolute value of 41)

Similarly to a negative value of program parameter, set a0 to -41 and the outcome is s0= 41, too.




**Assignment 2: Create a project to implement Home Assignment 2. Compile and simulate it. Change the program parameters (registers a0, a1, a2) and observe the execution results. Run the program in the single-step mode and pay attention to the changes in registers, especially the pc and ra registers.**

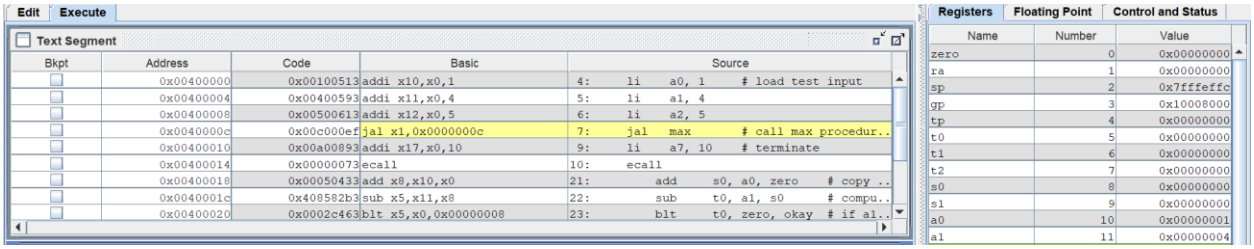
We change the value of program parameters a0, a1, a2 respectively to 1, 4, 5 as the picture below:

Edit Execute

exercise.asm*

```
1  # Laboratory Exercise 7, Home Assignment 2
2  .text
3  main:
4      li    a0, 1      # load test input
5      li    a1, 4
6      li    a2, 5
7      jal   max        # call max procedure
8
9      li    a7, 10     # terminate
10     ecall
11 end_main:
12
13 # ---------------------------------------------------------------------
14 # Procedure max: find the largest of three integers
15 # param[in]  a0  integers
16 # param[in]  a1  integers
17 # param[in]  a2  integers
18 # return     s0   the largest value
19 # ---------------------------------------------------------------------
20 max:
21         add     s0, a0, zero     # copy a0 in s0; largest so far
22         sub     t0, a1, s0       # compute a1 - s0
```

After executing the instruction “jal max”, value of register ra changes from 0x00000000 to 0x00400010 which is the address of the next instruction “li a7, 10”. The register pc jumps from address 0x0040000c to address 0x00400018

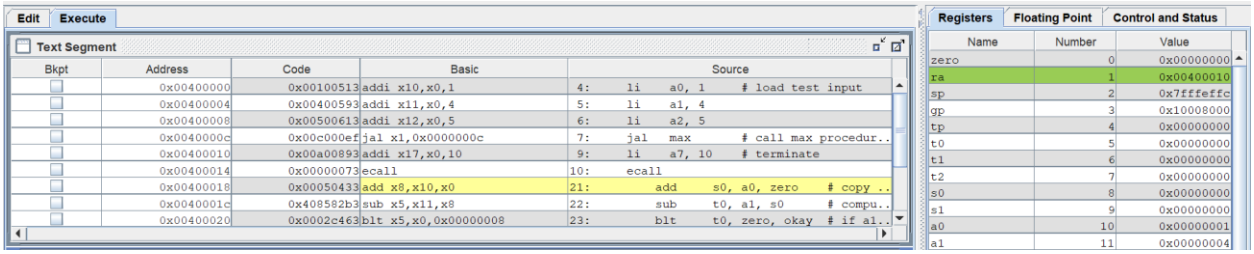


Text Segment

| Bkpt | Address | Code | Basic | Source |
|---|---|---|---|---|
| | 0x00400000 | 0x00100513 | addi x10,x0,1 | 4: li a0, 1 # load test input |
| | 0x00400004 | 0x00400593 | addi x11,x0,4 | 5: li a1, 4 |
| | 0x00400008 | 0x00500613 | addi x12,x0,5 | 6: li a2, 5 |
| | 0x0040000c | 0x00c000ef | jal x1,0x0000000c | 7: jal max # call max procedur.. |
| | 0x00400010 | 0x00a00893 | addi x17,x0,10 | 9: li a7, 10 # terminate |
| | 0x00400014 | 0x00000073 | ecall | 10: ecall |
| | 0x00400018 | 0x00050433 | add x8,x10,x0 | 21: add s0, a0, zero # copy .. |
| | 0x0040001c | 0x408582b3 | sub x5,x11,x8 | 22: sub t0, a1, s0 # compu.. |
| | 0x00400020 | 0x0002c463 | blt x5,x0,0x00000008 | 23: blt t0, zero, okay # if a1.. |

Registers | Floating Point | Control and Status

| Name | Number | Value |
|---|---|---|
| zero | 0 | 0x00000000 |
| ra | 1 | 0x00000000 |
| sp | 2 | 0x7fffeffc |
| gp | 3 | 0x10008000 |
| tp | 4 | 0x00000000 |
| t0 | 5 | 0x00000000 |
| t1 | 6 | 0x00000000 |
| t2 | 7 | 0x00000000 |
| s0 | 8 | 0x00000000 |
| s1 | 9 | 0x00000000 |
| a0 | 10 | 0x00000001 |
| a1 | 11 | 0x00000004 |



Text Segment

| Bkpt | Address | Code | Basic | Source |
|---|---|---|---|---|
| | 0x00400000 | 0x00100513 | addi x10,x0,1 | 4: li a0, 1 # load test input |
| | 0x00400004 | 0x00400593 | addi x11,x0,4 | 5: li a1, 4 |
| | 0x00400008 | 0x00500613 | addi x12,x0,5 | 6: li a2, 5 |
| | 0x0040000c | 0x00c000ef | jal x1,0x0000000c | 7: jal max # call max procedur.. |
| | 0x00400010 | 0x00a00893 | addi x17,x0,10 | 9: li a7, 10 # terminate |
| | 0x00400014 | 0x00000073 | ecall | 10: ecall |
| | 0x00400018 | 0x00050433 | add x8,x10,x0 | 21: add s0, a0, zero # copy .. |
| | 0x0040001c | 0x408582b3 | sub x5,x11,x8 | 22: sub t0, a1, s0 # compu.. |
| | 0x00400020 | 0x0002c463 | blt x5,x0,0x00000008 | 23: blt t0, zero, okay # if a1.. |

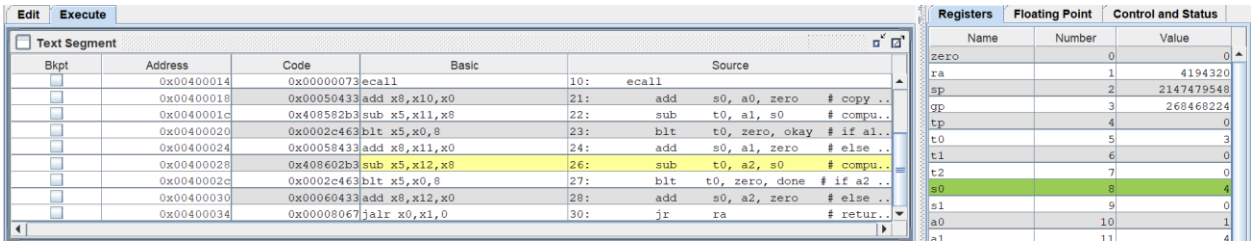
Registers | Floating Point | Control and Status

| Name | Number | Value |
|---|---|---|
| zero | 0 | 0x00000000 |
| ra | 1 | 0x00400010 |
| sp | 2 | 0x7fffeffc |
| gp | 3 | 0x10008000 |
| tp | 4 | 0x00000000 |
| t0 | 5 | 0x00000000 |
| t1 | 6 | 0x00000000 |
| t2 | 7 | 0x00000000 |
| s0 | 8 | 0x00000000 |
| s1 | 9 | 0x00000000 |
| a0 | 10 | 0x00000001 |
| a1 | 11 | 0x00000004 |

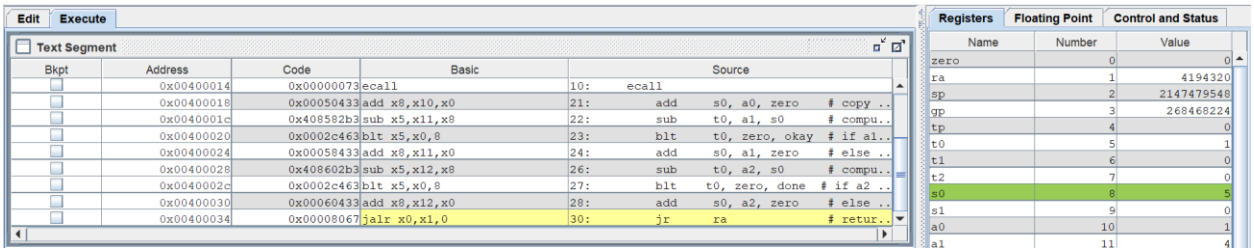
| | |
|---|---|
| pc | 0x0040000c |

| | |
|---|---|
| pc | 0x00400018 |

Because a1 (=4) - s0 (=1) > 0, then s0 is set to a1 or s0 = 4.

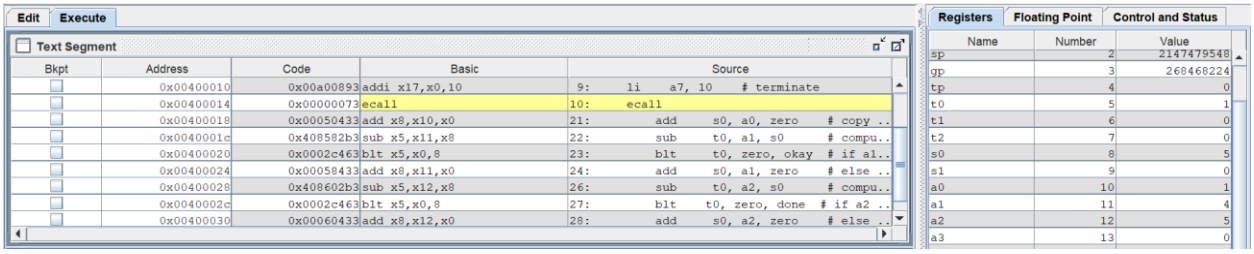



Again, a2 (=5) – s0 (=4) > 0, then s0 is set to a2 or s0 = 5




After executing the instruction “jr ra”, the register pc jumps from the address 0x00400034 to the address stored in the register ra which is 0x00400010


Then, the program ends and the result which is the max value among a0, a1, a2 (1, 4, 5) is 5 stored in the register s0. Correctly!!




**Assignment 3: Create a project to implement Home Assignment 3. Compile and simulate it. Change the program parameters (registers s0, s1), observe the process and results. Pay attention to changes in the sp register. Observe the memory pointed to by sp in the Data Segment window.**

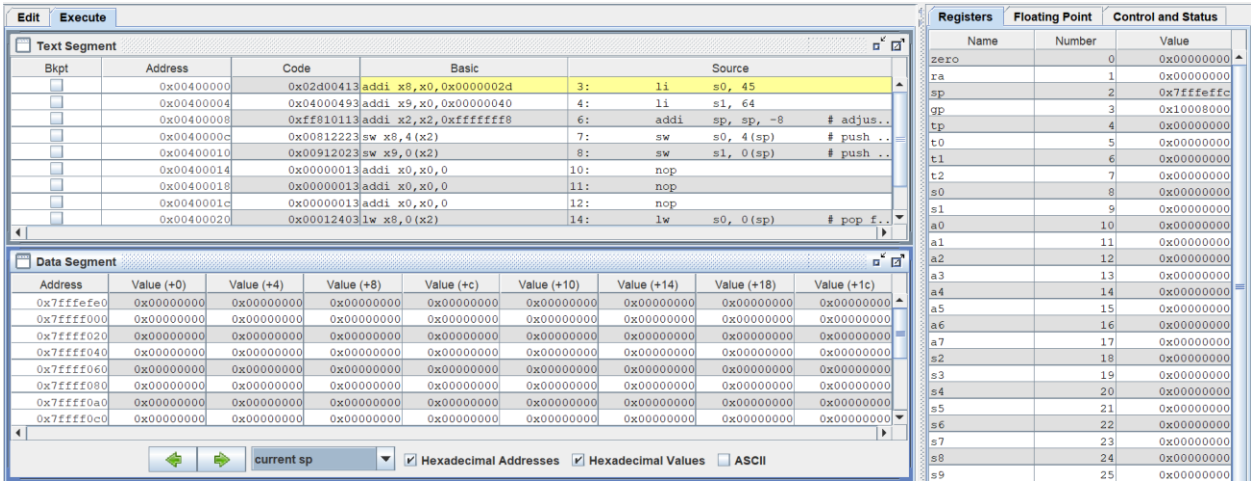
We change the program parameters (register s0, s1) that s0 = 45, s1 = 64

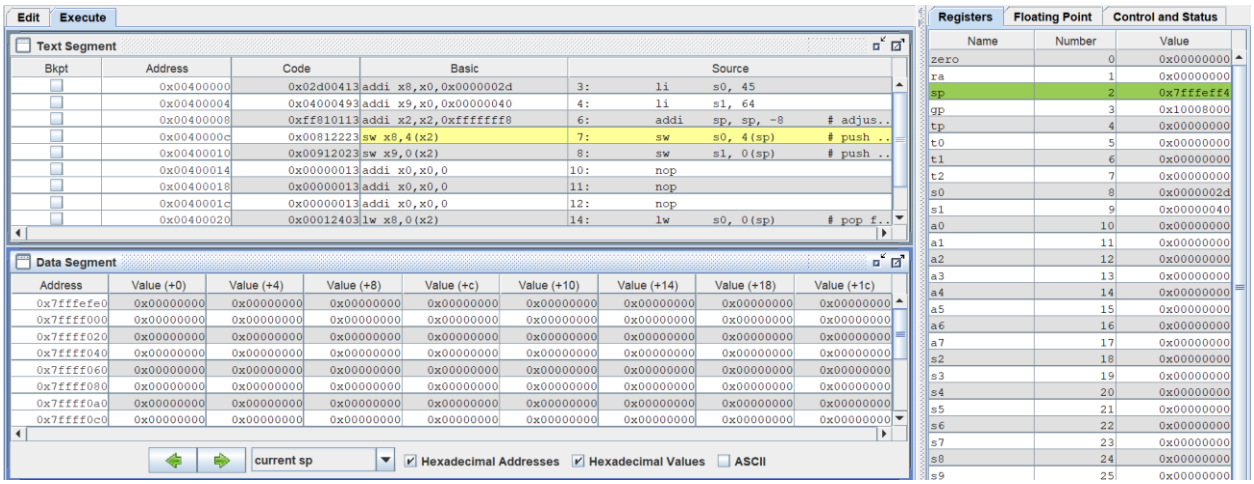
Edit | Execute

exercise.asm

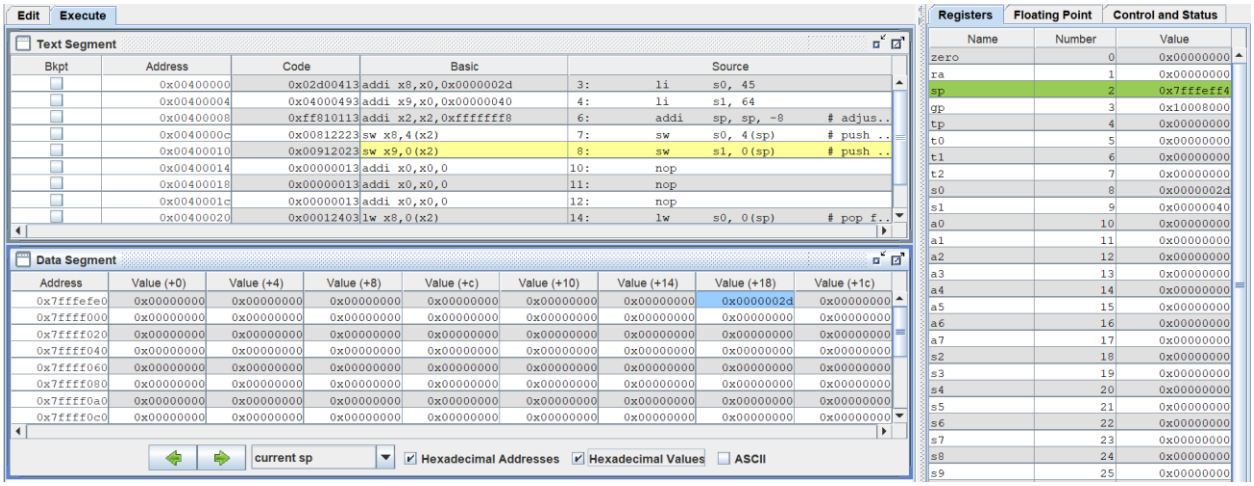
```
1  # Laboratory Exercise 7, Home Assignment 3
2  .text
3          li      s0, 45
4          li      s1, 64
5  push:
6          addi    sp, sp, -8      # adjust the stack pointer
7          sw      s0, 4(sp)       # push s0 to stack
8          sw      s1, 0(sp)       # push s1 to stack
9  work:
10         nop
11         nop
12         nop
13 pop:
14         lw      s0, 0(sp)       # pop from stack to s0
15         lw      s1, 4(sp)       # pop from stack to s1
16         addi    sp, sp, 8       # adjust the stack pointer
```

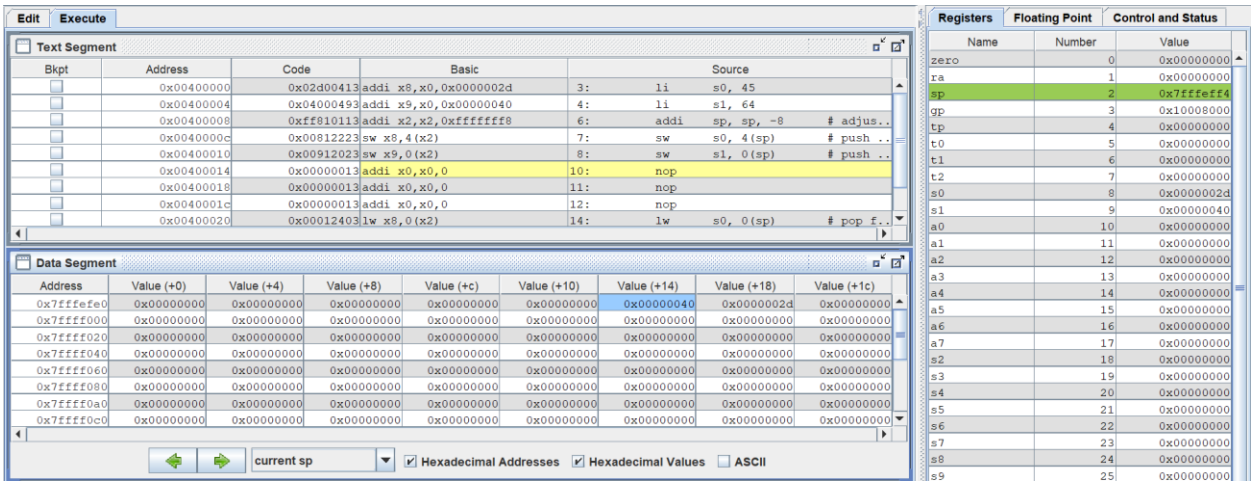
Initially, the register sp has value of 0x7fffeffc. After executing the instruction " addi sp, sp, -8", the register sp points to the address 0x7fffeff4



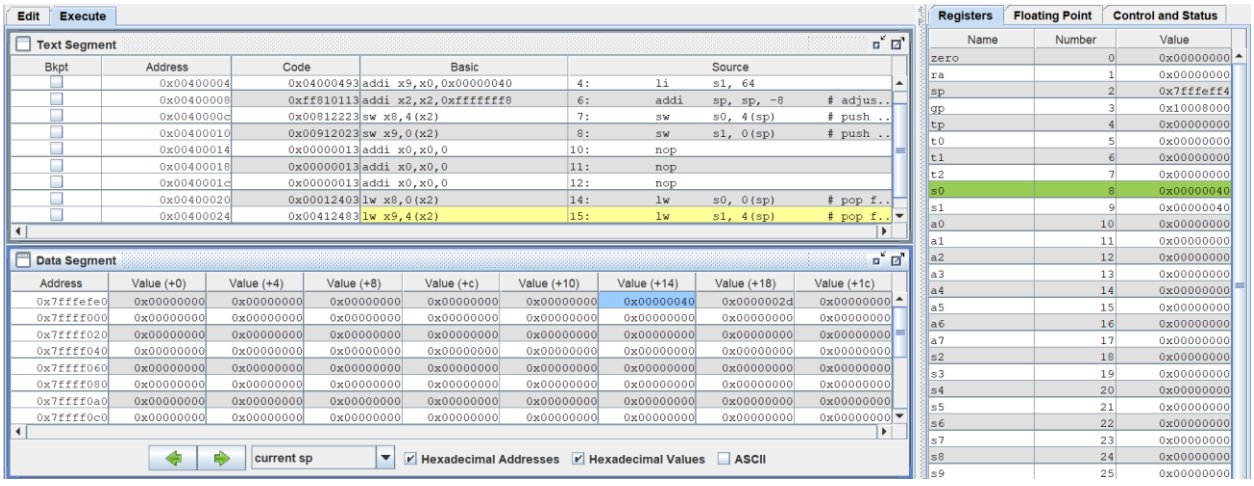



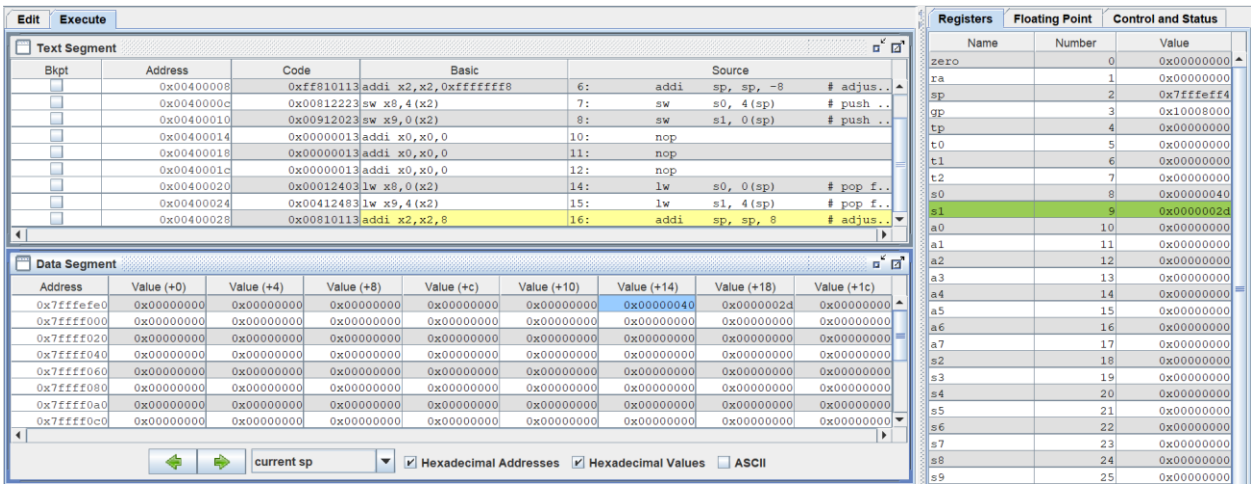
The value of s0, s1 is stored respectively in the address 0x7fffeff8 ( sp + 4) and 0x7fffeff4 (sp)



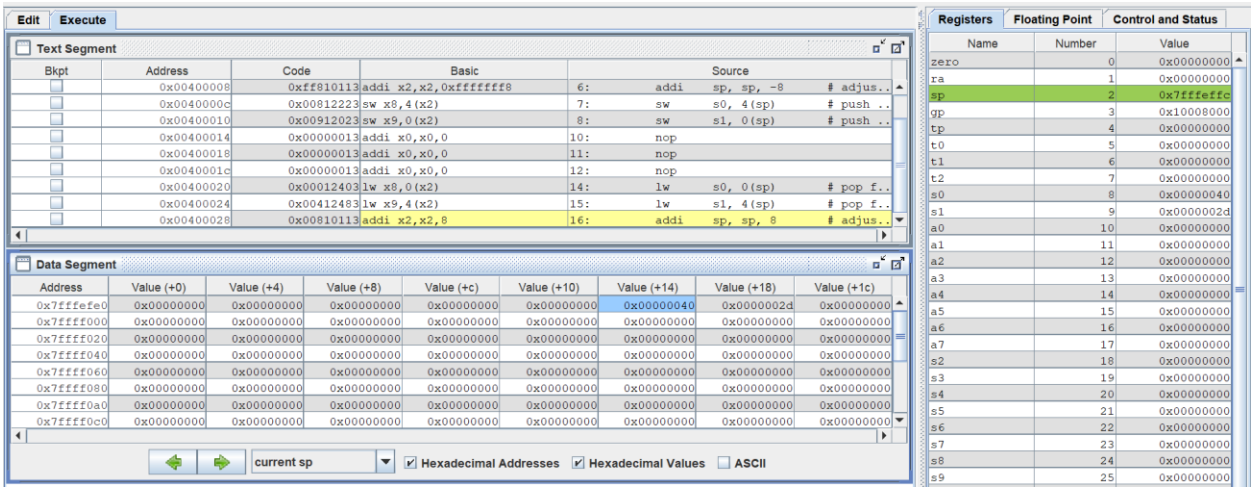




Then, loading the content in the address 0x7fffeff4 (sp) to s0, the content in the address 0x7fffeff8 (sp+4) to s1. At this time, s0 = 64, s1 = 45. They swapped the value.






After all, we free space in the register sp, return the allocated stack memory by restoring the original value of the sp register. Then, the program ends.

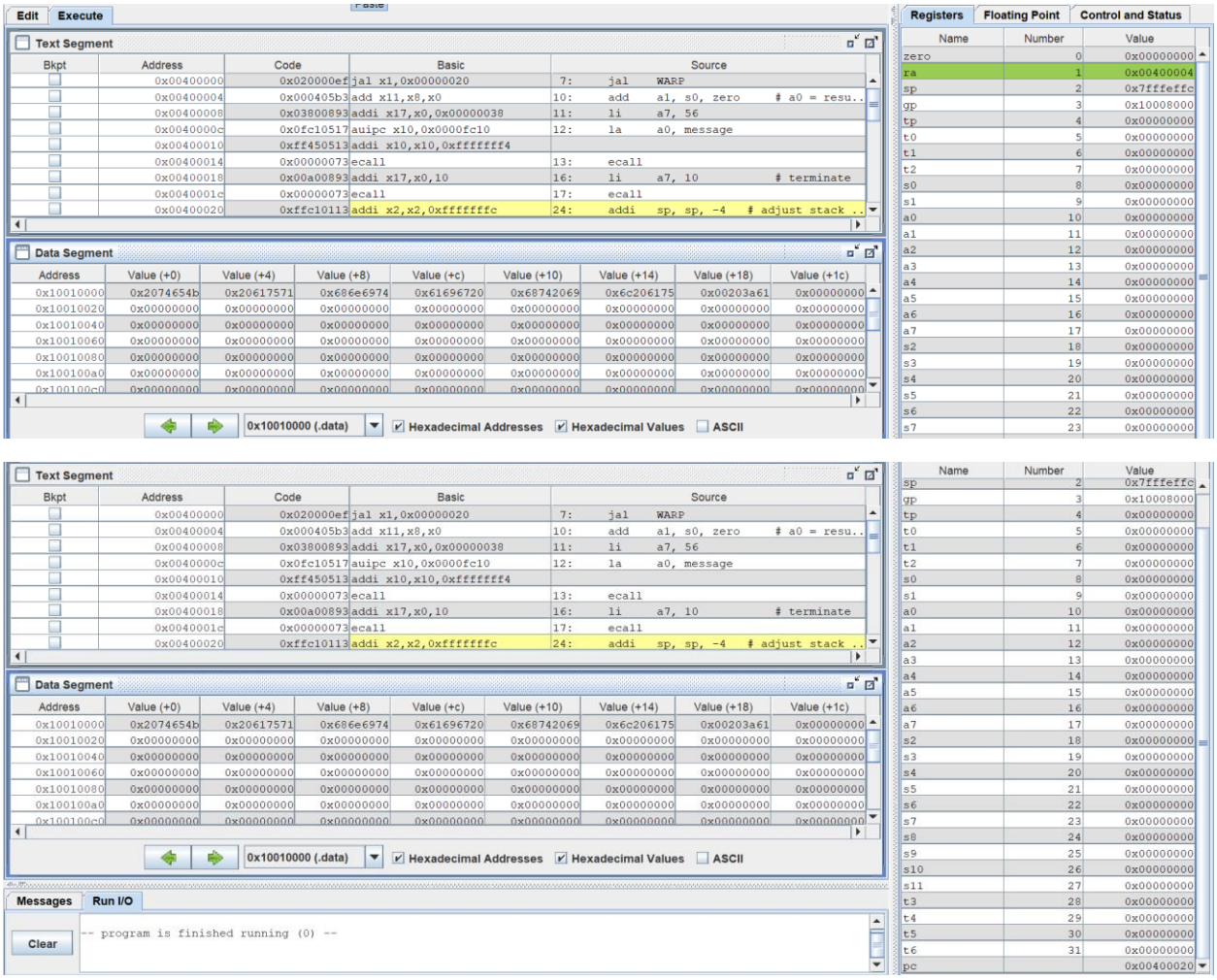



**Assignment 4: Create a project to implement Home Assignment 4. Compile and simulate it. Change the parameter in the a0 register and check the result in the s0 register. Run the program in the single-step mode and observe the changes in the registers pc, ra, sp, a0, s0. List the values in the stack memory when executing the program with n = 3.**

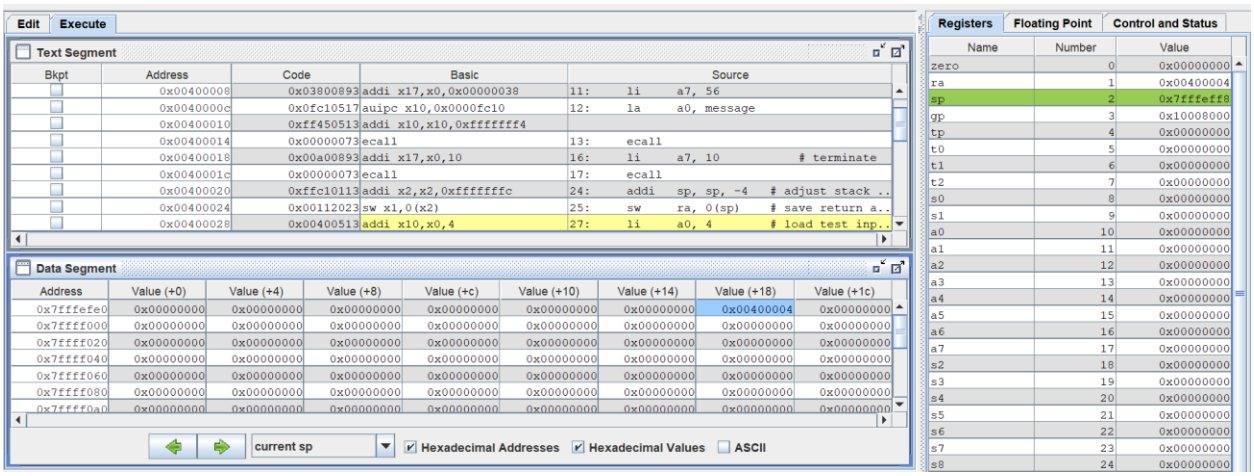
We change the value of parameter in the register a0 to 4 as the picture below:

```
19
20  # ------------------------------------------------------------------
21  # Procedure WARP: assign value and call FACT
22  # ------------------------------------------------------------------
23  WARP:
24      addi    sp, sp, -4    # adjust stack pointer
25      sw      ra, 0(sp)     # save return address
26
27      li      a0, 4         # load test input N
28      jal     FACT          # call fact procedure
29
30      lw      ra, 0(sp)     # restore return address
31      addi    sp, sp, 4     # return stack pointer
32      jr      ra
33  wrap_end:
34
```

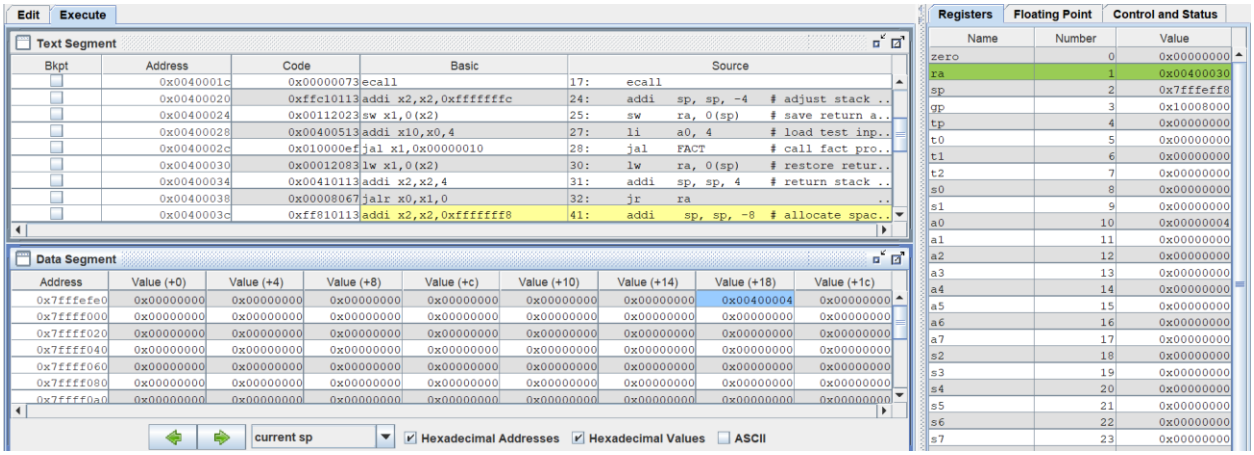
After executing the instruction " jal WARP", the program jumps to label WARP and the register ra stores the address of the instruction right after "jal WARP" which is ra = 0x00400004. The register pc jumps to address of label WARP



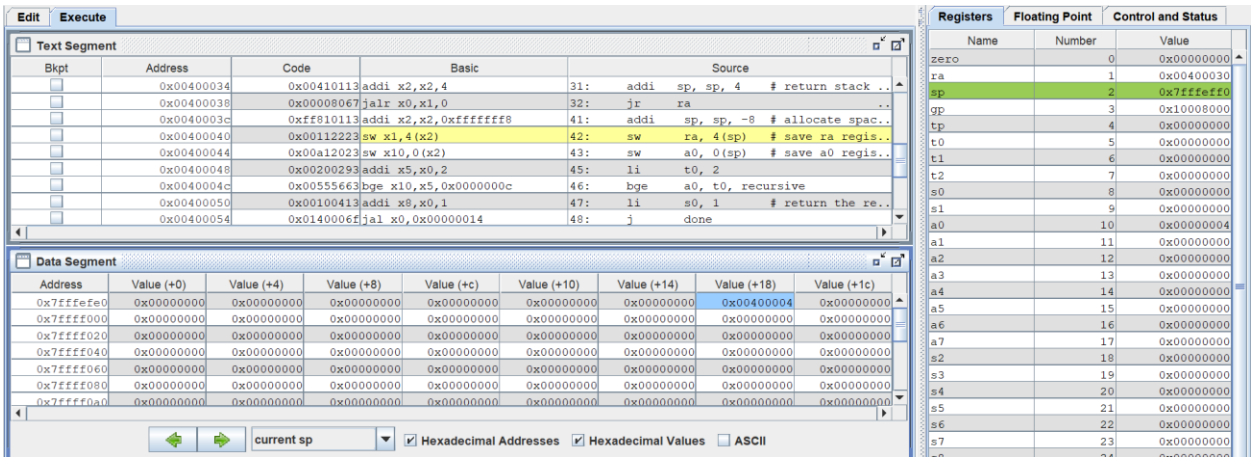

After that, the register sp is made a room for storing the address in register ra in order to mark the original place and be ready to execute the next subroutine. This address is stored in memory which the address is 0x7fffeff8. This process is for program coming back to execute the next instruction below the subroutine. It's like a calling a fuction in high programming language.

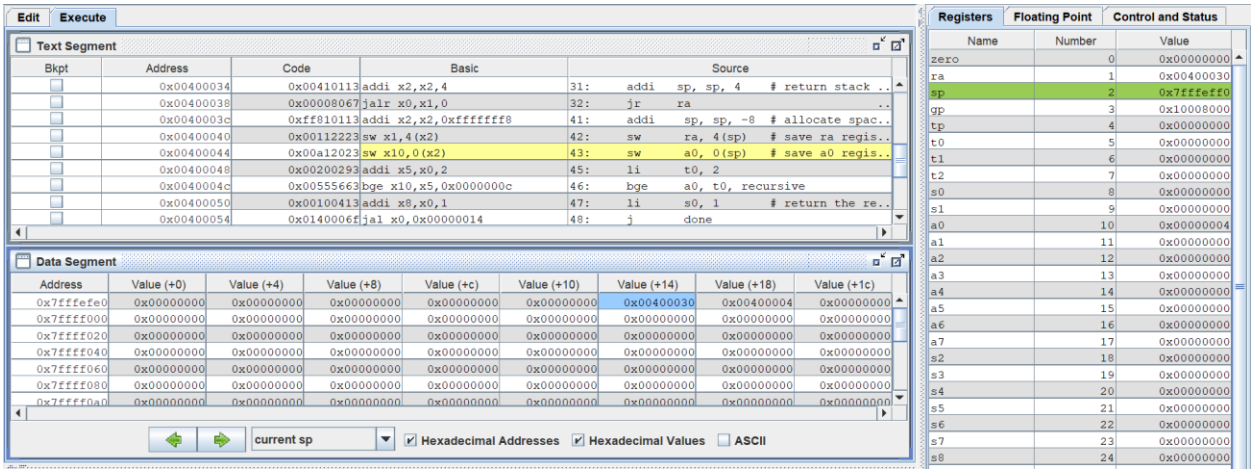


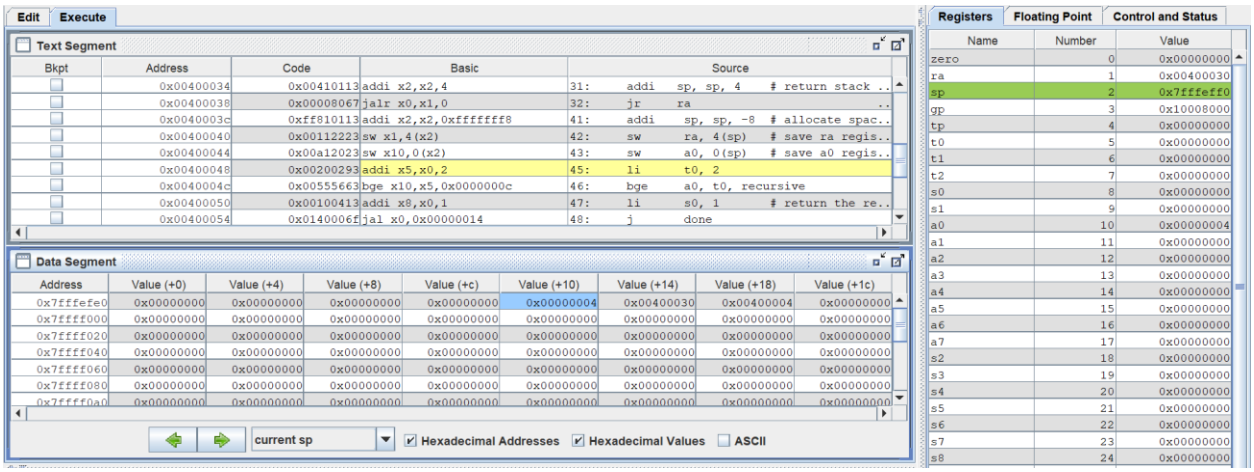

After executing the instruction " jal FACT", the pc jumps from address 0x0040002c to address 0x0040003c. The register ra stores the address of the instruction right after "jal FACT" which is 0x00400030.

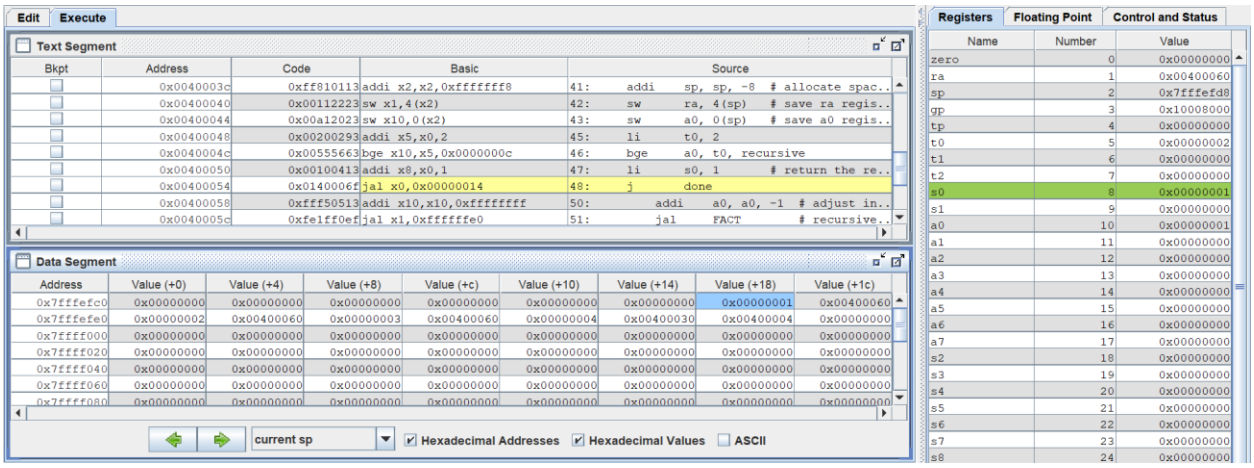

The stack is continually made space to store value of register ra and a0. It is important because this process helps the program could come back to the instruction right after the jump instruction called before, execute the rest instructions correctly. We have to store value of a0 in order to use a0 to compute without losing its original value during recursion subroutine takes place. This is because after each subroutine was done, we can load the initial value of a0 from memory in stack pointed by register sp.



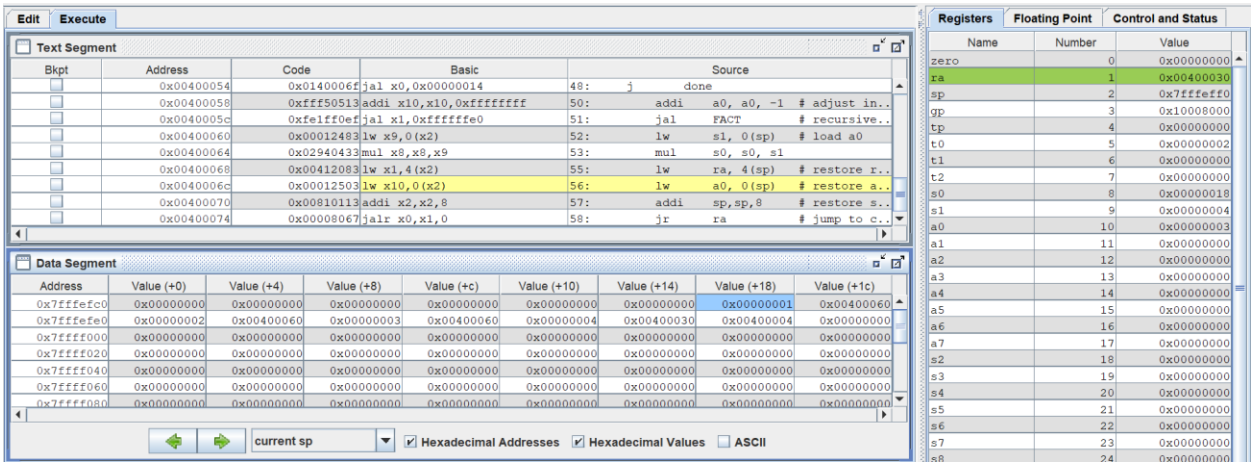

The recursion subroutine is called until a0 < 2 which is the base case and s0 storing the result is set to value of 1.



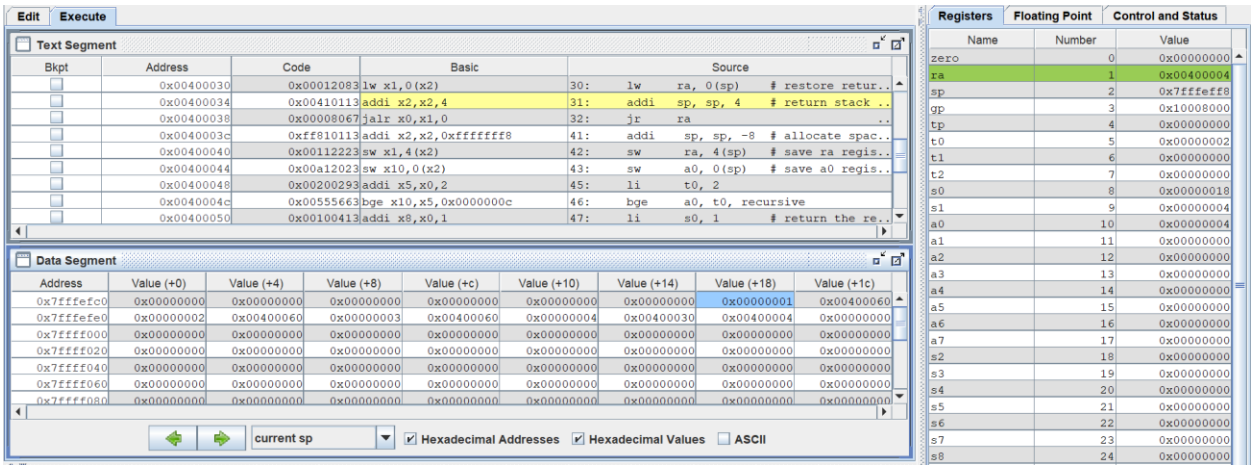

Then, the program jumps to label done, load the address in stack to register ra, which is the address of the instruction ” lw s1, 0(sp)  # load a0”. And execute the part which is computing factorial step-by-step. The register pc jumps to address stored in ra



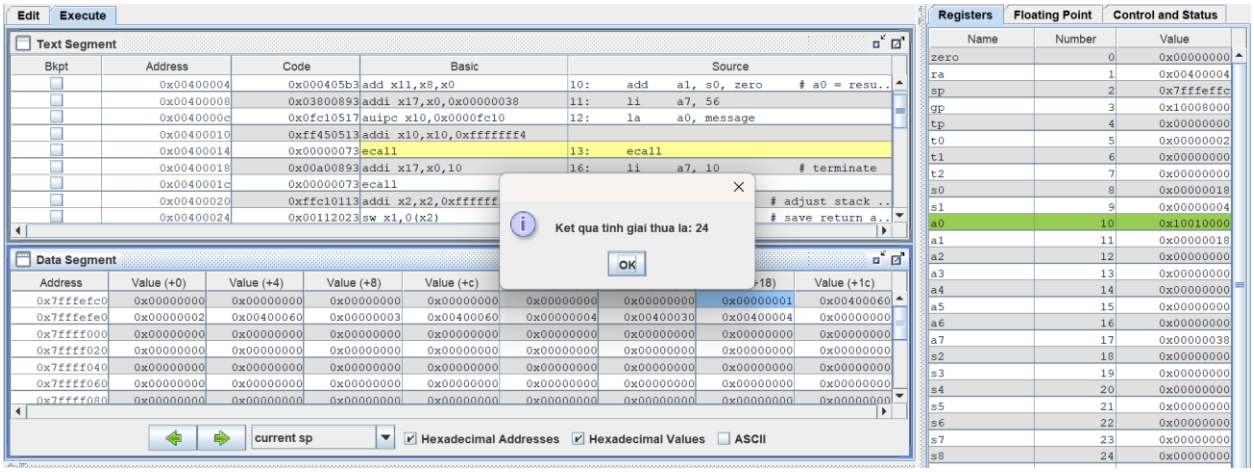
This process has gone on until ra has value 0x00400030 and program exits the subroutine FACT



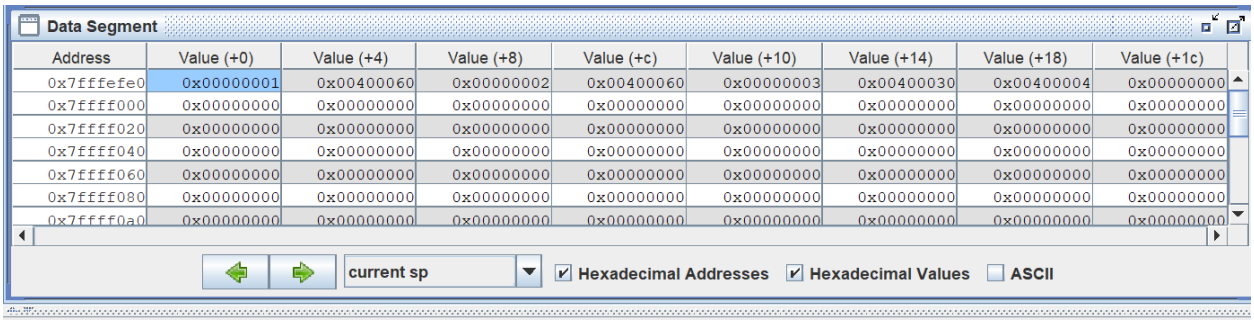

Then, register ra gets value of 0x00400004 and exits the label WARP. After each loading content from stack, sp is increased to remove space created in storing purpose.




After all, the program ends and register s0 = 1*2*3*4 or s0 = 24, print the output on dialog.




These are all values in stack with n=3:




**Assignment 5: Write a subroutine to find the largest value, the smallest value, and their respective positions in a list of 8 integers stored in the registers from a0 to a7. For example:**

- **Largest: 9, 3 → The largest value is 9, stored in a3.**
- **Smallest: -3, 6 → The smallest value is -3, stored in a6.**

**Hint: Use the stack memory to pass parameters.**

The source code is as below

Edit Execute

lab 7.asm

```
1  # Laboratory Exercise 7, Assignment 5
2  .data
3        mess1: .asciz  "The largest value is: "
4        mess2: .asciz "The smallest value is: "
5  .text
6  main:
7      add t0, sp, zero
8      li t3, 1
9      li a0, -3
10     li a1, 2
11     li a2, -6
12     li a3, 4
13     li a4, 10
14     li a5, 7
15     li a6, -1
16     li a7, 1
17     addi sp, sp, -28
18     sw a1, 0(sp)
19     sw a2, 4(sp)
20     sw a3, 8(sp)
21     sw a4, 12(sp)
22     sw a5, 16(sp)
```

Edit Execute

lab 7.asm

```
22     sw a5, 16(sp)
23     sw a6, 20(sp)
24     sw a7, 24(sp)
25     li t1, 0
26     li t2, 0
27     add s0, a0, zero
28     add s1, a0, zero
29     jal    find
30
31 print:
32     li      a7, 4
33     la      a0, mess1
34     ecall
35     li a7, 1
36     add a0, s1, zero
37     ecall
38     li a7, 11
39     li a0, 44
40     ecall
41     li a7, 11
42     li a0, 32
43     ecall
```

Edit Execute

lab 7.asm

```
43      ecall
44      li a7, 1
45      add a0, t2, zero
46      ecall
47      li a7, 11
48      li a0, 10
49      ecall
50      li a7, 4
51      la a0, mess2
52      ecall
53      li a7, 1
54      add a0, s0, zero
55      ecall
56      li a7, 11
57      li a0, 44
58      ecall
59      li a7, 11
60      li a0, 32
61      ecall
62      li a7, 1
63      add a0, t1, zero
64      ecall
```

Edit Execute

lab 7.asm

```
64      ecall
65      li a7, 11
66      li a0, 10
67      ecall
68  quit:
69      li      a7, 10          # terminate
70      ecall
71  end_main:
72  find:
73      beq sp, t0, done
74      lw a1, 0(sp)
75      addi sp, sp 4
76      addi t3, t3, 1
77  min:
78      blt s0,  a1, max
79      add s0, a1, zero
80      addi t1, t3, -1
81  max:
82      bgt s1, a1, find
83      add s1, a1, zero
84      addi t2, t3, -1
85      j find
```

```
85      j find
86  done:
87          jr ra
```

The result is:

Data Segment

| Address | Value (+0) | Value (+4) | Value (+8) | Value (+c) | Value (+10) | Value (+14) | Value (+18) | Value (+1c) |
|---|---|---|---|---|---|---|---|---|
| 0x7fffefe0 | 2 | -6 | 4 | 10 | 7 | -1 | 1 | 0 |
| 0x7ffff000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x7ffff020 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x7ffff040 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x7ffff060 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x7ffff080 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x7ffff0a0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

current sp ☑ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

Messages Run I/O

Clear

```
The largest value is: 10, 4
The smallest value is: -6, 2
```